PC tree tensor data model speedups

Brett Viren

April 18, 2024

Problem

Converting from `PointCloud::Tree::Points` / `NaryTree::Node` and `ITensor` representations is "too slow".

Background concepts:

- "PC tree" is the working representation and "tensor data model" is "generic" representation.
- The **tensor data model** covers most WCT data types (`IFrame`, `ICluster`, `Dataset/Array`).
- We have other data models / formats for serializing some WCT data types.
- The "data model" is somewhat independent from "file format".
  - Eg, TDM now can serialize to JSON+Numpy, future may be mapped to HDF5

Test and summary results

```
$ ./build/aux/check_tensordm_pctree_speed
```

- Makes PC tree as root node with 1M children nodes.
- Each child has ≈ 75 point `(x,y,z,q)` point cloud.

Timing before and after some speedups:

before	after	step
5871	5814	made pctree
15957	3202	convert to tensordm
11146	0	indexed tensors
21539	3115	convert to pctree

Times are in millisecond, $\approx 15\%$ uncertainty for conversions (rerun variance).

Some reasons for the slowness

Based on testing with Google PerfTools.

- Some reference/howto slides on PerfTools later.

Original "tensor data model" for PC tree produced many small tensors

- $n_{tensors} = n_{nodes} \times n_{pcs} \times n_{arrays} + n_{bookkeeping}$
- Dataset/Array metadata object (`JsonCPP::Value`) ctor/dtor dominates.

$\Rightarrow$ The overhead of many small objects adds up.

Currently, we translate blob-sampled points and their attributes.

- Really only need to translate far smaller set of blob-level info.

$\Rightarrow$ Reducing will help.

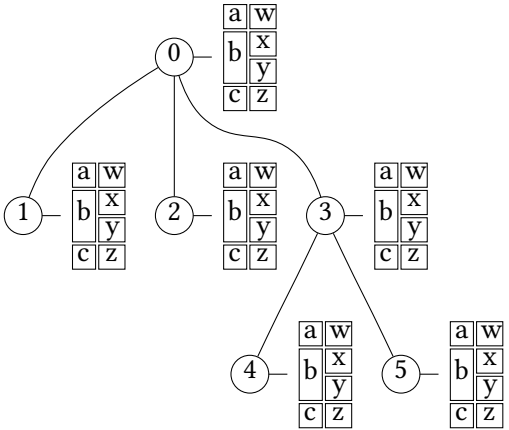
Speed up 1 - lazy metadata

Make `Dataset` hold metdata as `std::unique_ptr<JsonCPP::Value>`

- Was originally held by value.
- Avoids apparently costly ctro/dtor when no metatdata is required.
- Only a minor speed up as most tensors need non-zero metadata.

Leads to idea to change representation to reduce number of tensors....

Speedup 2 - new tensor data model for PC tree



Flatten the tree using a **parentage map**

- number nodes in DFS order

node index:	0	1	2	3	4	5
parent index:	0	0	0	0	3	3

Concatenate node-local PCs

- record per-node, per-PC sizes

	0	1	2	3	4	5
a	n_0	0	0	0	0	0
b	0	n_1	n_2	n_3	0	0
c	0	0	0	0	n_4	n_5

Changes to PointCloudDataset and Array

Add methods:

```
Dataset Dataset::slice(size_t pos, size_t len) const;
Dataset Dataset::slice(size_t pos, size_t len, bool share);
Array Array::slice(size_t pos, size_t len) const;
Array Array::slice(size_t pos, size_t len, bool share);
```

These allow to split up the concatenated PCs back to node-local PCs.

- They are now responsible for about 60% of `as_pctree()`.

How to CPU-profile with Google PerfTools

```
$ export PROG=./build/aux/check_tensordm_pctree_speed  # short hand
$ export CPUPROFILE=cpu.prof                           # required

$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libprofiler.so.0 $PROG [args]

$ google-pprof --pdf $PROG $CPUPROFILE > cpu.pdf
$ google-pprof --focus 'as_pctree' --pdf $PROG $CPUPROFILE > cpu.pdf

$ google-pprof --help
```

Many options, per file, per line.

How to HEAP-profile with Google PerfTools

```
$ export PROG=./build/aux/check_tensordm_pctree_speed  # short hand
$ export HEAPPROFILE=mem.prof                          # required

$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libtcmalloc.so.4 $PROG
```

Same PDF render commands as for CPU, replace `$CPUPROFILE` file with `$HEAPPROFILE`.

- Heap profiling greatly slows down running (unlike CPU profiling).
- If impatient, intermediate files like `mem.prof.NNNN.heap` can be used.
- Intermediate results may of course false-indicate leaks.

Still to do

Reconfigure / recode so we convert only blob-level info, drop sampled points.